Distributed Data Analytics Framework for Smart Transportation

Alexander J. Howard*, Tim Lee*, Sara Mahar*, Paul Intrevado, Diane Myung-kyung Woodbridge
{ajhoward7,tdlee,semahar2,pintrevado,dwoodbridge}@usfca.edu
Data Science Program
University of San Francisco

***Abstract*—As the amount data from IoT devices on transportation systems increases, developing a robust pipeline to stream, store and process data became critical. In this study, We explore prediction accuracy and computational performance of various supervised and unsupervised algorithms perform on distributed systems for developing a smart transportation data pipeline. Using a subset of New York City Taxi & Limousine Commission data, we evaluate Logistic Regression, Random Forrest Regressors and Classifiers, Principal Component Analysis, and Gradient Boosted Regression and Classification Tree machine learning techniques on a commodity computer as well as on a distributed system. Employing Amazon S3, EC2 and EMR, MongoDB, and Spark, we identify the conditions—data size and algorithm—under which the performance of distributed systems excel.**

***Index Terms*—Distributed computing, Distributed information systems, Machine learning, Smart transportation**

I. Introduction

Large cities in the United States such as Los Angeles and New York City suffer from chronic traffic congestion. In 2017, drivers in New York—the third most congested city in the world—spent an average of 91 hours in traffic. Considering travel time, operating costs, excess fuel, and lost productivity, the costs associated with traffic congestion in New York City alone are estimated at $20 billion annually [1]. In an effort to mitigate some of the effects of congestion, various studies have evaluated transportation data to analyze travel patterns, providing real-time predictions for drivers and passengers [2][3].

There have been significant investments in Internet of Things (IoT) devices and systems aimed at improving transportation systems, including traffic monitoring, live location streaming, and vehicle performance monitoring. With the recent expansion of rideshare services and autonomous vehicles, the volume of traffic data has increased rapidly. Verizon recently announced 40% growth in IoT network connections for transportation systems in 2017 [4]. Developing a robust pipeline to stream, store and analyze real-time data is a critical piece of infrastructure for IoT devices.

In order to reliably ingest and process transportation data from multiple users in real-time, it is important to schedule and process data efficiently, while avoiding system failures. Distributed systems utilize many commodity computers connected via network, providing high-quality aggregate performance. This is achieved by dividing a task into multiple smaller tasks and processing each task on one or multiple machines. While an individual high-performance machine also provides efficient data processing and task scheduling, it is more susceptible to failure [5]. Distributed systems are not only cost-efficient and fast, but are also reliable. By replicating data across multiple nodes, the system is resilient against the failure of a single node.

As transportation data is collected from heterogeneous sources that are likely to evolve over time, highly-flexible NoSQL (Not Only SQL) database solutions are becoming more prominent [6]. Moreover, NoSQL databases are designed to run on a cluster in an effort to maximize throughput. MongoDB is one of the most popular NoSQL databases [7][8].

As the volume of data increases, MongoDB outperforms relational databases for data insertion, update, deletion and other operations. For instance, inserting 1,000,000 records takes 57,871 milliseconds for MongoDB, and takes 882,078 milliseconds for an Oracle database. Deleting 1,000,000 records only takes 1 millisecond for MongoDB, and takes 38,079 milliseconds for an Oracle database [9]. Employing MongoDB, we can achieve time and cost efficiencies for storing and querying large volumes of transportation data.

The open-source Apache Spark [10] processing engine provides a comprehensive framework to perform the aforementioned tasks in a distributed setting, i.e., across a number of different machines. Spark provides an interface via Python, R, Scala or Java to program parallelized clusters with fault tolerance. The architecture is such that a central master node distributes data and individual processes to a family of worker nodes. With large datasets, this parallelization provides incredible performance boosts by concurrently running tasks. Included within the Spark stack are functionalities to perform SQL-style database queries, employ built-in machine learning pipelines and process streaming data.

This research examines how Spark performance responds to varying both the type of machine learning algorithm employed and the size of dataset, with the objective of determining the conditions under which Spark excels. We seek to provide a benchmark for those seeking to balance algorithmic precision and data size, and identifying the conditions under which it becomes beneficial to implement distributed systems for machine learning (ML) algorithms. We compare the performance of distributed systems with similar processes being run on a single laptop with PySpark, Pandas and Scikit-learn, which are

* These authors contributed equally.

a Spark Python API, a data manipulation and analysis library, and a machine learning library for Python, respectively.

We use New York City Taxi data [11], a publicly available dataset that details every trip taken within NYC. As this dataset was the subject of a 2017 Kaggle Competition [12], we have the added benefit of comparing our results with top Kaggle competitors, whose results often considered state of the art.

II. Background

Hadoop's MapReduce software, introduced in 2004, was the first to implement distributed techniques in an attempt to speed up large scale data analysis [13]. MapReduce splits data into smaller chunks across different nodes, and subsequently maps and processes a task, e.g., filtering and sorting, in parallel. The output of a mapped task becomes the input of a reduce operation, which performs a summary operation. This highly-effective model allows users to design programs with successive Map and Reduce operations, and is still in production use today.

Spark was designed in UC Berkeley's AMPLab in 2009 and open-sourced in 2010. Although Spark adopts MapReduce concepts, it runs up to 100 times faster than Hadoop MapReduce, utilizing in-memory computing and an advanced task-execution engine [14]. Spark also has built-in libraries that allow for efficient iterative computation. Included with version 0.8 in 2013, MLLib was the first library within Spark to support Machine Learning features. The MLLib API worked with Sparks native Resilient Distributed Datasets (RDD)—a fundamental data structure of Spark—and has since been placed into maintenance in favor of Spark ML [15]. Although MLLib currently provides greater functionality, Spark ML provides a more user-friendly API with objects stored as DataFrames, a spreadsheet-like collection of data organized in columns. We employ Spark ML for all subsequent analysis.

There is a paucity of academic research that benchmarks Spark ML performance, either evaluating the library itself under varying conditions, or in comparison to MLLib. A number of researchers from DataBricks authored a 2016 paper providing an academic introduction to MLLib [16], and discussing its performance across versions. This research provides a similarly extensive look at Spark ML performance, examining results generated when running Spark ML on a specific dataset. The interested reader is directed to the following references, which provide a deep analysis of this data:

- Fischer-Baum [17] – The authors analyzed the New York City Taxi & Limousine Commission (NYC TLC) data from January 2009 through June 2015, as well as Uber data during that same period, concluding Uber is taking millions of rides away from taxis in Manhattan.
- Schneider [18] – The author analyzed pickup and drop-off locations using NYC taxi and Uber data, containing 19 million rides for the periods from April to September 2014 and from January to June 2015. Highly insightful visualizations are provided, depicting customer demand and routing.
- Daulton's Harvard Data Science Project [19] – The author used Spark to pre-process data, Pandas and Scikit-learn to apply Machine Learning techniques to predict drop-off locations, and generated several visualizations.

III. System Overview

A. System Workflow

For this research, the data science pipeline was designed around scalability, cloud resources and distributed methods. Technologies were selected to build an ingestion and prediction system for taxi data from multiple cities and resources. We therefore selected Amazon Web Services (AWS) as the primary platform to host storage, data extraction, transform and load (ETL) processes and machine learning tasks. The exact hardware specifications of the database and testing cluster are noted in Figure 1 1.

1) **Data Storage**: Data scraped using APIs and Selenium was stored in Amazon's Simple Storage Service (S3). S3 was selected for its replicated hosting in multiple data-centers, as well as ease of interoperability with Amazon's cloud servers [20].
2) **Data Management**: When manipulating and analyzing taxi data from varied geographic locations, data format consistency is not guaranteed. Although relational database service (RDS) platforms can be efficiently and reliably run on a single, high-performance instance, they often run into difficulties with data heterogeneity and large volumes of data. We therefore selected MongoDB, a flexible NoSQL platform, based on its sharding (the ability to natively and automatically spread data across an arbitrary number of servers) and replication capabilities. We hosted an AWS EC2 cluster (`t2.xlarge instance`) running MongoDB.[21].
3) **Data Analysis**: Apache Spark is a popular distributed computing software platform built on Java. Although similar to Hadoop MapReduce, Spark is optimized to work in memory, thereby increasing data processing performance. Spark also includes its own distributed versions of popular machine learning algorithms. Spark's distributed data processing jobs can be managed by Spark itself, or by other cluster managers. We chose to use YARN (Yet Another Resource Negotiator) as our cluster manager, primarily for it's ease of integration with AWS EMR (Elastic Map Reduce) services [22]. EMR automatically provisions hardware resources, installs the required software and provides an accessible monitoring dashboard. For this research, we ran three `m4.2xlarge` instances, with one instance set up as a master and the remaining two as slaves. For comparison, additional experiments were performed one standard commercial laptop with SSD storage, 8GB RAM, and 2.9 GHZ Intel Core i5.

B. Algorithms

Spark ML—the Spark machine learning library—only supports algorithms whose performance improves when dis-

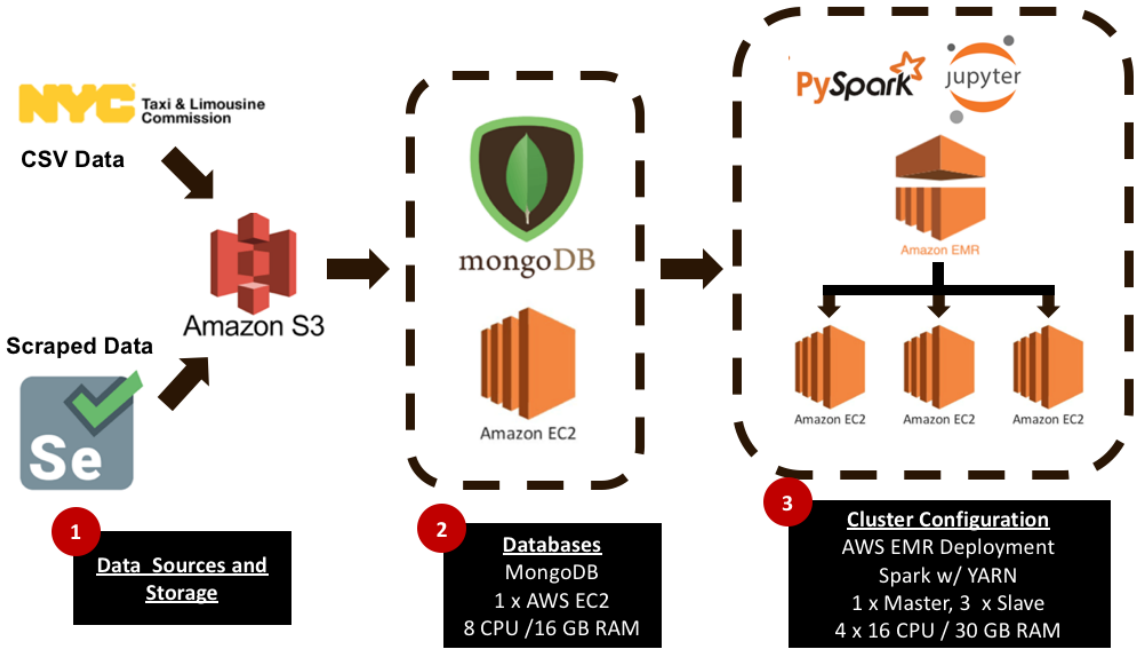

Fig. 1: System workflow

tributed across a cluster. The implication of this carefully circumscribed algorithmic selection is that some of the most popular machine learning algorithms are not implemented. Algorithms that tend to work poorly within the distributed framework rely on boosting techniques, i.e., iteratively combining weak learners to form a single stronger learner. This process requires a large amount of data-shuffling between individual nodes in a cluster, which is detrimental to performance. In contrast, bagging is a technique where learners are constructed independently and results averaged. This independent nature allows several learners to be trained simultaneously across different nodes within the cluster, improving overall performance.

In light of this, we conducted our analysis on a broad spectrum of supervised and unsupervised learning techniques, including Random Forest Regressors, Random Forest Classifiers and Principal Component Analysis (PCA). To establish a baseline for comparison, we also tested Logistic Regression and K-means clustering algorithms, which are less computationally expensive than the tree ensembles of a Random Forest, allowing us to accurately characterize the additional efficiencies provided by a distributed setting.

1) Random Forests: A Random Forest is an ensemble of decision trees, where the output is either a mean prediction (regression setting) or the mode of the classes (classification setting) of the constituent trees. Each individual tree is exposed to a (potentially bootstrapped) subset of the rows and columns and consists of a series of binary splits that look to optimize the loss function. This inherent randomness within the trees avoids overfitting issues complicit with deterministic decision trees [23]. The compartmentalized nature of the algorithm with independent trees is especially well-suited to the distributed framework, and is therefore an excellent candidate for this research.

Random Forests have become an increasingly popular technique that has the added benefit of requiring a minimal amount of architecture and hyper-parameter tuning, making them relatively easy to train on a dataset. Tree depth is an important hyper-parameter of a random forest algorithm, which we tune and evaluate in this research. Although increasing tree depth improves a model's predictive accuracy, it requires a longer training time. Moreover, it may cause overfitting, a scenario where the model is overly sensitive to training data and insufficiently sensitive to test data. Spark ML's API provides a method to specify the maximal depth of any individual tree within the forest, allowing us to analyze the relationship between tree depth and the size of data used to train the random forest.

Another benefit of using Random Forests is ease of interpretability. We can easily quantify the feature importance in any model by observing the effect that randomizing each feature has on model prediction quality. We explore this facet by testing our model with questions to which we already know the answer, and cross-reference our results with the answers we know are correct. Additional details of this validation approach follow in the appropriate subsections.

a) Random Forest Regression: Using a previous Kaggle competition as our guide [12], we use Random Forest Regressors to predict trip duration from the remaining features. Although this analysis is conducted under mildly different conditions to those who competed in the Kaggle competition,

we demonstrate that our algorithms are performing at a level comparable to the state of the art.

b) Random Forest Classification: We examine a classification problem by attempting to predict the type of taxi used on a given trip based on the features of the trip. This is in many ways a trivial problem: New Yorkers will inform you that green taxis *only* pick up passengers from outer boroughs, whereas yellow taxis can pick up passengers from any location, including Manhattan. One can imagine that pick-up location alone would act as a very strong predictor of the color. We therefore use this problem to evaluate how the accuracy and feature importance of our predictions change with the size of data evaluated in a Random Forest Classifier.

2) Logistic Regression: Traditional Logistic Regression is a binomial or multinomial classification algorithm [24]. Logistic Regression minimizes, through gradient descent, a linear combination of our input features when passed through a logit function [25]. Although Logistic Regression is a traditional statistical classification method developed in the 1950s, it remains highly utilized and will serve as a a baseline for the aforementioned taxi color-prediction classification problem solved with a Random Forest Classifier.

3) K-means Clustering: Perhaps the most popular clustering algorithm, K-means clustering seeks to partition n observations into K distinct clusters. After a random cluster initialization, the algorithm iteratively assigns each point to the closest centroid of existing clusters [26]. We did not generate particularly strong results from the methodology given the size and density of the dataset. Rather, we examined how the computational time and distribution of nodes in each cluster changes as the the size of the dataset is varied.

4) Principal Component Analysis: Principal Component Analysis (PCA) is a tool primarily used for exploratory data analysis and feature engineering. PCA seeks to reduce the dimensionality of a large dataset by easily extracting and identifying salient features into a subset of independent linear combinations. In finding each principal component, the algorithm selects a new axis where variance is maximized [27].

5) Gradient Boosted Trees for Regression / Classification: Gradient Boosted Trees are similar to their Random Forest counterparts. They benefit from a collection of decision trees, subsequently making a prediction based on the weighted scoring from each of those trees [28]. The primary difference in Gradient Boosted Trees is that the first tree is used to make a prediction, and, once evaluated, an additional tree is added such that it minimizes error, i.e., minimizes the loss of the first tree. Trees are added, one at a time, each minimizing the loss of the preceding tree, until a robust model is developed. As trees are added sequentially and not in parallel—owing to the dependency of earlier tree predictions—this reduces the performance benefit of distributed computing. For our purposes, Gradient Boosted Tree Regressors and Classifiers were tested on smaller datasets to benchmark the processing time along with the accuracy.

IV. Experiment Output

A. Data

The New York City Taxi & Limousine Commission (TLC) publishes monthly data that records select features of every recorded journey within the city [11]. The data is segmented into Yellow taxis, Green taxis and contains fields such as pick-up and drop-off locations, trip times, trip distances, payment types and number of passengers. We additionally supplemented the existing data with Manhattan weather conditions, obtained from Weather Underground using stylized code [29].

In our research, we selected a subset of 51 million observations, collected between January and May of 2017. For Yellow and Green taxis, the size of monthly data was 850MB and 90MB, respectively.

As our objective was to benchmark performance rather than achieve marginal gains in prediction accuracy, we conducted minimal feature engineering, although greater attention to detail in this area would almost certainly improve predictive performance. We did, however, parse the dates to extract day of week, day of month, month and year information. We also removed all observations with trip durations that exceeded two hours.

B. Results

The Root Mean Squared Error (RMSE) is a typical metric by which predictions are evaluated, computed as the root of the squared sum of distances of predicted and true values. For the *k*-th record, the error in predicting taxi route distance in miles is calculated. Then all errors over *n* total number of records is averaged and squared rooted to calculate RMSE. (see Equation 1).

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(predicted_k - true_k)^2} \qquad (1)$$

Random Forest Regressor results compare well with Kaggle competition winners. When using a Random Forest with 2 million observations, we obtain a RMSE on the validation set of 0.22, which is comparable with Kaggle's winning RMSE 0.29 with a training set of 1.5 million rows. Random Forest Classifiers took up to 3 minutes to train, and the Random Forest Regressors took up to 6 minutes to train, with maximal tree depth parameters less than 18 (Figure 2).

System Architecture: Launching a Spark cluster is not an efficient architecture for small datasets, owing to the additional computational cost of running a distributed system. Our experiments demonstrate that one does not begin to benefit from reduced training time of Random Forests until the dataset consists of at least 1 million rows (Figure 3). At this point,

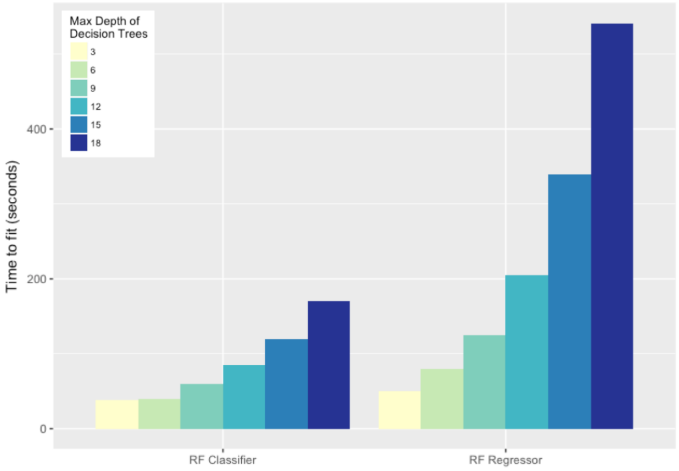

Fig. 2: Random Forest training times varying tree depth.

however, it quickly becomes infeasible to train the model on a single laptop with Pandas and Scikit-learn. Training a Random Forest with Pandas and Scikit-learn on 3 million rows generates memory errors, therefore direct comparisons of training times are not possible for large datasets.

The Spark cluster, however, is capable of dealing with datasets of any size; one is always able to increase the memory of individual nodes and/or increase the total number of nodes to distribute computational load. There is no theoretical limit to the improvements in training times you can achieve from scaling the cluster. The only true restriction on computational power is the need to shuffle data and establish worker nodes.

Under all test conditions in our experiments, there existed none which it was optimal to run PySpark on a single laptop. The overhead computational costs associated with parallelizing data across the laptop cores and subsequently shuffling it are far greater than running the optimized Scikit-learn Random Forest algorithm. The PySpark setup also quickly runs into memory errors, crashing with datasets larger than 100k rows.

Algorithmic Comparisons: When testing supervised machine-learning methods (Random Forest, Logistic Regression), we observed spikes in training times (Figure 4). This is attributed to running into a computational ceiling, owing to both the complexity of the algorithm and the volume of data. This combination forces Spark to shuffle data or temporarily write to disk, significantly increasing computational time. Random Forest Regression hit this processing ceiling at much smaller data sizes, whereas Logistic Regression can process larger amount of data. This unwanted behavior can be mitigated with higher-performance nodes or an increasing the quantity of nodes. As a result, it is recommended that distributed hardware decisions be informed by an understanding of an algorithm's complexity.

When employing a Spark cluster, our experiments demonstrate that the difference in training times between unsupervised algorithms (K-means and PCA) can be orders of magnitude faster than Random Forests (Figure 4). Even within

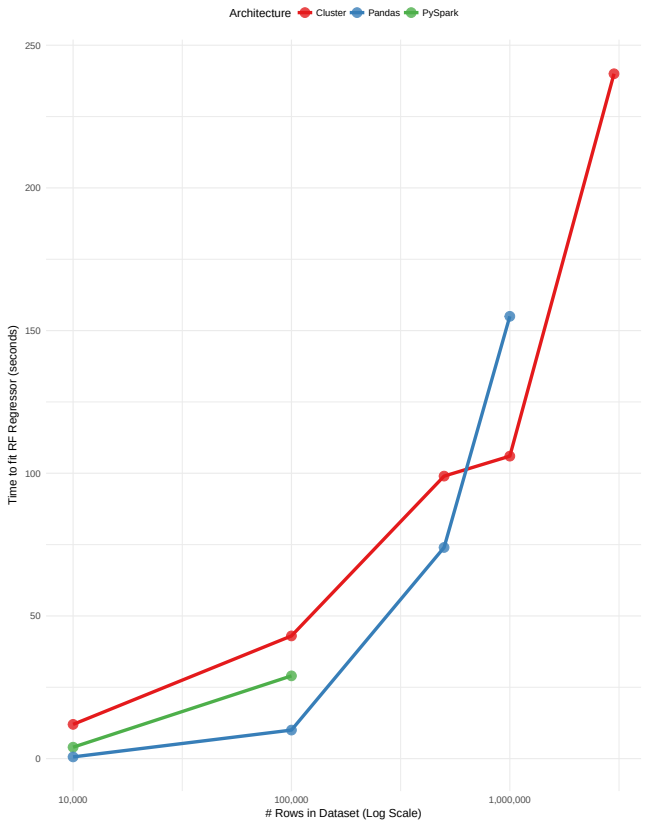

Fig. 3: Comparison of Random Forest Regressor fit times for cluster vs PySpark (standalone) vs Pandas.

between classification algorithms, the Logistic Regression baseline model takes less than half the time to train than a Random Forest Classifier, and is 10 times faster than a Gradient Boosted Tree Classifier (Figure 5). The performance gap between Logistic and Random Forest methods can be modulated by tuning parameters such as tree depth, sampling and number of trees. This result suggests one should consider a more conservative algorithmic approach when operating on a distributed framework: start with a simple algorithm, and *incrementally* increase the algorithmic complexity.

Moreover, the faster Logistic Regression algorithm also demonstrates superior predictive power to its tree ensembles counterparts (Figure 5). These results will not generalize to all problems, but it does highlight the need to be judicious with modeling choices: complex models will not necessarily outperform less complex ones. The modeler must choose the most appropriate model, with considerations for hypothesis, scope, training time, infrastructure, budget and data.

V. Conclusion

The combination of low-cost cloud computing and Spark's easy-to-use interface provide a powerful resource for ML practitioners to create competitive large scale models. For ML problems such as NYC taxis with large volumes (trips) and high cardinality (locations), Spark's scalable throughput

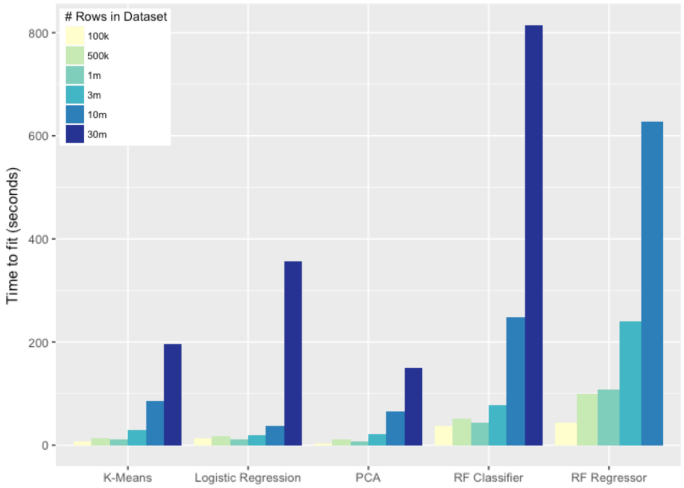

Fig. 4: Comparison of fit times on the cluster for each algorithm and dataset size.

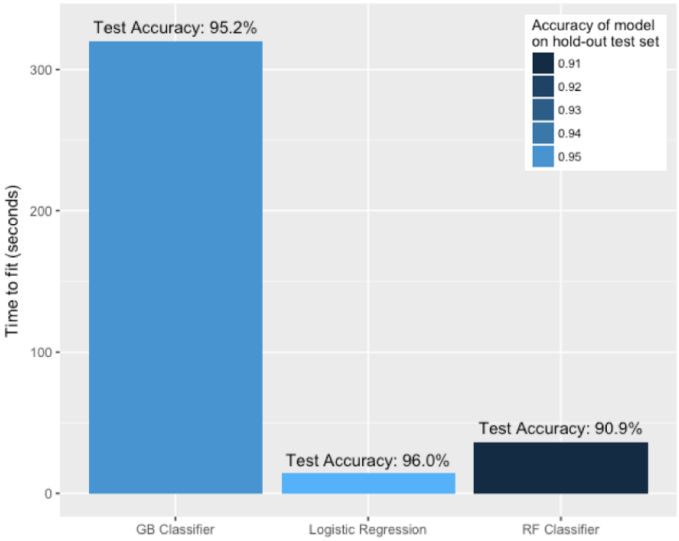

Fig. 5: Comparison of accuracy and fit times for 3 core classification algorithms - 100k rows.

enables ML teams to avoid common big data compromises such as reducing problem scope, widening predictive accuracy requirements, or avoiding the application of complex ML techniques. In addition, Spark's pre-built ML algorithm library enables a smooth transition of smaller models into larger scale production; Spark ML's data structures, concepts, and parameters are consistent with other popular python ML libraries.

Our research concluded that a Spark cluster was unquestionably a good choice to run machine learning algorithms at scale, though this result does come with a number of caveats. Practitioners must carefully weigh the trade-off between improvements in predictive performance and run-time (cost) for their particular application. With this particular dataset, we found that not only did increasing the dataset size have a negligible improvement on our classification accuracy, but also that using complex tree-ensemble methods did little to improve results generated from simpler algorithms. In fact, the simpler Logistic Regression algorithm had a higher accuracy on the hold-out test set than either of the more complex algorithms.

With this in mind, one has to be careful in choosing their methods. Data size nor algorithmic complexity will alone maximize prediction accuracy. In particular, if a marginal improvements in predictability are not being sought, then a simpler method—perhaps one that also has greater interpretability—may be a more appropriate machine learning algorithm.

As the results of this research are based on a single dataset, it would be instructive to analyze a range of different datasets—both content and size—to compare runtimes and predictive performance relative to state of the art results.

References

[1] Partnership for New York City. (2018, January) $100 billion cost of traffic congestion in metro new york. [Online]. Available: http://pfnyc.org/wp-content/uploads/2018/01/2018-01-Congestion-Pricing.pdf

[2] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva, "Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2149–2158, 2013.

[3] R. Gerte, K. C. Konduri, and N. Eluru, "Is there a limit to adoption of dynamic ridesharing systems? evidence from analysis of uber demand data from new york city," Tech. Rep., 2018.

[4] Verizon. (2017) State of the market: Internet of things 2017. [Online]. Available: http://www.verizon.com/about/sites/default/files/Verizon-2017-State-of-the-Market-IoT-Report.pdf

[5] D. M.-k. Woodbridge, A. T. Wilson, M. D. Rintoul, and R. H. Goldstein, "Time series discord detection in medical data using a parallel relational database," in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1420–1426.

[6] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in *Proceedings of the 51st ACM Southeast Conference*. ACM, 2013, p. 5.

[7] MongoDB. (2018) Mongodb for giant ideas. [Online]. Available: https://www.mongodb.com/

[8] K. Chodorow, *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.", 2013.

[9] A. Boicea, F. Radulescu, and L. I. Agapin, "Mongodb vs oracle–database comparison," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*. IEEE, 2012, pp. 330–335.

[10] Apache Spark, "Apache spark: Lightning-fast cluster computing," 2018. [Online]. Available: http://spark.apache.org

[11] NYC Taxi & Limousine Commission. (2018) Tlc trip record data. [Online]. Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[12] Kaggle. (2017) New york city taxi trip duration. [Online]. Available: https://www.kaggle.com/c/nyc-taxi-trip-duration

[13] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] L. Gu and H. Li, "Memory or time: Performance evaluation for iterative operation on hadoop and spark," in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 721–727.

[15] Apache Spark. (2018) Ml pipelines. [Online]. Available: https://spark.apache.org/docs/2.2.0/ml-pipeline.html

[16] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[17] R. Fischer-Baum and C. Bialik, "Uber is taking millions of manhattan rides away from taxis," *FiveThirtyEight Economics*, 2015.

[18] T. Schneider, “Analyzing 1.1 billion nyc taxi and uber trips, with a vengeance,” 2015.
[19] S. Daulton. (2015) Nyc taxi data prediction. http://sdaulton.github.io/TaxiPrediction/.
[20] Amazon Web Service . (2018) Amazon s3. [Online]. Available: https://aws.amazon.com/s3/
[21] Amazon Web Service. (2018) Amazon ec2. [Online]. Available: https://aws.amazon.com/ec2/
[22] Amazon Web Service. (2018) Overview of amazon emr architecture. [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview-arch.html
[23] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[24] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman, *Applied linear statistical models*. Irwin Chicago, 1996, vol. 4.
[25] J. S. Cramer, “The origins and development of the logit model,” *Logit models from economics and other fields*, vol. 2003, pp. 1–19, 2003.
[26] J. A. Hartigan and M. A. Wong, “Algorithm as 136: A k-means clustering algorithm,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.
[27] J. Shlens, “A tutorial on principal component analysis,” *arXiv preprint arXiv:1404.1100*, 2014.
[28] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, “Boosting algorithms as gradient descent,” in *Advances in neural information processing systems*, 2000, pp. 512–518.
[29] Weather Underground. (2018) Historical weather. [Online]. Available: https://www.wunderground.com/history/